Development and Testing of a Vehicle Management System for Autonomous Spacecraft Habitat Operations

G. Aaseng [*] and J. Frank [†] and M. Iatauro [‡] and C. Knight [§] and R. Levinson [¶] and J. Ossenfort[‖] M. Scott [**] and A. Sweet [††]
NASA Ames Research Center, Moffett Field, CA 94035-1000

J. Csank [‡‡] and J. Soeder[§§]
NASA Glenn Research Center, 21000 Brookpark Rd, Cleveland, OH 44135

D. Carrejo[¶¶] and A. Loveless [***]and T. Ngo[†††]
NASA Johnson Space Center, 2101 E NASA Pkwy, Houston, TX 77058

Z. Greenwood[‡‡‡]
NASA Marshall Space Flight Center, Redstone Arsenal Huntsville, AL 35812

As the increased distance between Earth-based mission control and the spacecraft results in increasing communication delays, small crews cannot take on all functions performed by ground today, and so vehicles must be more automated to reduce the crew workload for such missions. In addition, both near-term and future missions will feature significant periods when crew is not present, meaning the vehicles will need to operate themselves autonomously. NASA's Advanced Exploration Systems Program pioneers new approaches for rapidly developing prototype systems, demonstrating key capabilities, and validating operational concepts for future human missions beyond low-Earth orbit. Under this program, NASA has developed and demonstrated multiple technologies to enable the autonomous operation of a dormant space habitat. These technologies included a fault-tolerant avionics architecture, novel spacecraft power system and power system controller, and autonomy software to control the habitat.

The demonstration involved simulation of the habitat and multiple spacecraft sub-systems (power storage and distribution, avionics, and air-side life-support) during a multi-day test at NASA's Johnson Space Center. The foundation of the demonstration was 'quiescent operations' of a habitat during a 55 minute eclipse period. For this demonstration, the spacecraft power distribution system and air-side life support system were simulated at a high level of fidelity; additional systems were managed, but with lower fidelity operational constraints and system behavior. Operational constraints for real and simulated loads were developed by analyzing on-orbit hardware and evaluating future Exploration capable technology. A total of 13 real and simulated loads were used during the test. Eight scenarios including both nominal and off-nominal conditions were performed. Over the course of the test, every application performed its desired functions successfully during the simulated tests. The results will inform both future tests, as well as provide insight to NASA's domestic and international partners, as they

[*]Computer Scientist, Intelligent Systems Division, M/S/269, NASA Ames Research Center, AIAA Member.
[†]Principal Investigator, Autonomous Systems and Operations Project, Intelligent Systems Division, M/S/269, NASA Ames Research Center, AIAA Non-Member.
[‡]Computer Scientist, Intelligent Systems Division, M/S/269, NASA Ames Research Center, AIAA Non-Member.
[§]Computer Scientist, Intelligent Systems Division, M/S/269, NASA Ames Research Center, AIAA Non-Member.
[¶]Computer Scientist, Intelligent Systems Division, M/S/269, NASA Ames Research Center, AIAA Non-Member.
[‖]Computer Scientist, Intelligent Systems Division, M/S/269, NASA Ames Research Center, AIAA Non-Member.
[**]Computer Scientist, Intelligent Systems Division, M/S/269, NASA Ames Research Center, AIAA Non-Member.
[††]Computer Engineer, Intelligent Systems Division, M/S 269-1, NASA Ames Research Center, AIAA Non-Member.
[‡‡]Electrical Engineer, Power Management and Distribution, M/S 301-5, 21000 Brookpark Rd, NASA Glenn Research Center, AIAA Sr. Member.
[§§]Senior Technologist for Power, Power Division, M/S 301-5, 21000 Brookpark Rd, NASA Glenn Research Center, AIAA Non-Member.
[¶¶]Engineer, Systems Engineering and Test Branch, 2101 NASA Parkway, NASA Johnson Space Center, AIAA Non-Member.
[***]Network Engineer, Command and Data Handling Branch, 2101 NASA Parkway, NASA Johnson Space Center, AIAA Non-Member.
[†††]Engineer, Spacecraft Software Engineering Branch, 2101 NASA Parkway, NASA Johnson Space Center, AIAA Non-Member.
[‡‡‡]Aerospace Engineer, Environmental Control and Life Support Systems Branch, ES62, NASA Marshall Spaceflight Center, AIAA Non-Member.

construct the next generation of space habitats to be used on beyond-Earth missions.

$ACAWS$	=	Advanced Caution and Warning System
AES	=	Advanced Exploration Systems
$AMPS$	=	AES Modular Power System
APC	=	Autonomous Power Controller
API	=	Application Program Interface
CH_4	=	Methane
$CCCA$	=	Common Cabin Air Assembly
$CCDD$	=	cFS Command and Data Dictionary
cFS	=	core Flight System
$COTS$	=	Commercial Off The Shelf
CO_2	=	Carbon Dioxide
DE	=	Diagnostic Executive
$DIMA$	=	Distributed Integrated Modular Architecture
DoD	=	(Battery) Depth of Discharge
DS	=	Data Store
DS_REPLAY	=	Data Store Replay
$ECLSS$	=	Environmental Control and Life Support Systems
EVA	=	Extra-Vehicular Activity
EXP	=	Expediting Processing of Experiment for the Space Station Rack
FC	=	Flight Computer
FD	=	Fault Detector
FIR	=	Fault Impacts Reasoner
$FPGA$	=	Field Programmable Gate Array
$HyDE$	=	Hybrid Diagnosis Engine
H_2	=	Hydrogen
$iPAS$	=	Integrated Power Avionics and Software
Kw	=	Kilowatt
KwH	=	Kilowatt-Hours
$LOP-G$	=	Lunar Outpost-Gateway
LAN	=	Local Area Network
LC	=	Limit Checker
$L1V$	=	Level 1 Voter
$MBSE$	=	Model-Based Systems Engineering
$MBSU$	=	Main Bus Switching Unit
$NASA$	=	National Aeronautics and Space Administration
OGA	=	Oxygen Generation Assembly
$OSAL$	=	Operating System Abstraction Layer
O_2	=	Oxygen
PDU	=	Power Distribution Unit
$PLEXIL$	=	Plan Execution Interchange Language
PPA	=	Plasma Pyrolysis Assembly
PPE	=	Power-Propulsion Element
PWD	=	Potable Water Dispenser
RIU	=	Remote Interface Unit
RPC	=	Remote Power Controller
SAB	=	Sabatier Reactor
SAM	=	Spacecraft Atmosphere Monitor
SBN	=	Software Bus Network
$SBN-LIB$	=	Software Bus Network Library
$SCH-TT$	=	Scheduler - Time Triggered
$SCIP$	=	Solving Constrained Integer Programs
$SNRF$	=	Space Network Research Federation

TCP	=	Transmission Control Protocol
$TEAMS-RT$	=	Testability Engineering and Maintenance System (Real Time) (TEAMS-RT)
TT	=	Time-Triggered
TTE	=	Time-Triggered Ethernet
UDP	=	User Datagram Protocol
VDC	=	Volt Direct Current
VSM	=	Vehicle Systems Manager

I. Introduction

The United States National Aeronautics and Space Administration (NASA) is working with domestic and international partners to solve the great challenges of deep space exploration. Missions in the vicinity of the Moon will span multiple phases as part of NASA's framework to build a flexible, reusable and sustainable infrastructure that will last multiple decades and support missions of increasing complexity. This first phase of exploration near the Moon will use current technologies, enabling NASA to develop new techniques and apply innovative approaches to solving problems in preparation for longer-duration missions far from Earth.

NASA plans to construct a habitable spacecraft, currently referred to as the Lunar Orbital Platform-Gateway (LOP-G) [1], in the vicinity of the Moon. LOP-G consists of a Habitat, Airlock, Power and Propulsion Element (PPE), and Logistics module. LOP-G will support up to 4 crew for 30 days. The PPE will provide orbital maintenance, attitude control, communications with Earth, space-to-space communications, and radio frequency relay capability in support of extravehicular activity (EVA) communications. The Habitat provides habitable volume and short-duration life support functions for crew in cislunar space, docking ports, attach points for external robotics, external science and technology payloads or rendezvous sensors, and accommodations for crew exercise, science/utilization and stowage. The Airlock provides capability to enable astronaut EVAs as well as the potential to accommodate docking of additional elements, observation ports, or a science utilization airlock. A Logistics module will deliver cargo to the Gateway.

Of significant importance for these future missions is the balance between crew autonomy and vehicle automation. As noted above, small crews cannot take on all functions performed by ground today, and so vehicles must be more automated to reduce the crew workload for such missions. In addition, both near-term and future missions will feature significant periods when crew is not present, meaning the vehicles will need to operate themselves autonomously. A thorough assessment of human spaceflight dormancy and autonomy requirements for a future Mars mission describes the wide range of challenges to be faced [2]. The need for autonomy for the LOP-G is reflected in the Gateway concept of operations [1] and its requirements, as well as requirements for the first of its components, the PPE [3]. A number of projects have demonstrated different elements of human spaceflight autonomous mission operations, including crew autonomy [4], [5] i.e. enabling astronauts to operate their spacecraft without assistance from Earth-based Mission Control, and vehicle autonomy [6], [7], i.e. enabling the vehicle to operate on its own, whether crew are present or not. However, many of these demonstrations were either performed onboard the International Space Station (ISS), on older hardware and flight software, or in low fidelity analog environments. The LOP-G will have a different design, operating environment, and mission, and can take advantage of new flight software and avionics technology, thus motivating further technology development and demonstration activity.

NASA's Advanced Exploration Systems (AES) program[*] pioneers new approaches for rapidly developing prototype systems, demonstrating key capabilities, and validating operational concepts for future human missions beyond low-Earth orbit. Under this program, NASA has developed and tested multiple technologies to enable the autonomous operation of a dormant space habitat. These technologies included a fault-tolerant avionics architecture, novel spacecraft power system and power system controller, and autonomy software to control the habitat. The work in this paper continues the effort described in [8] to develop and demonstrate autonomy technology using contemporary flight software and automated reasoning technology. These technologies are described briefly below, and in more detail in the remainder of the paper.

The Habitat's Avionics Architecture consists of three Flight Computers (FCs), each running a real-time operating system and integrated with each other and other spacecraft subsystems over a Time-Triggered Ethernet (TTE) network. The use of a time-triggered (TT) paradigm enables the delivery of messages with low latency and constant jitter. Redundant switches are used to help ensure successful message delivery in the presence of faults. The flight software, including all subsystem control functions, data management, avionics bus messaging, etc. is based on NASA's Core

[*]https://www.nasa.gov/content/aes-overview

Flight System (cFS) framework. A time-triggered scheduler couples the flight software schedule to the network time base, and special cFS applications provide abstractions for messaging between different processors. The FCs run identical copies of the flight software, and commands from the FCs are resolved to protect against corrupted data.

The AES Modular Power System (AMPS) power storage and distribution system consists of two 120 Volt Direct Current (VDC) Main Bus Switching Units (MBSUs), cross-strapped to two Power Distribution Units (PDUs), which power independent loads. The AMPS hardware is managed by an Autonomous Power Controller (APC), which is responsible for ensuring the safe operation of the hardware, including fault management energy forecasting, and analyzing and implementing user load schedules which prioritize the loads for power.

Habitat autonomy technology is designed to manage the Habitat. It consists of multiple cFS applications performing spacecraft-wide operations and management functions. One set of applications is able to schedule Habitat subsystems during nominal operations and execute those schedules autonomously. Additional applications detect faults that reduce available energy to spacecraft subsystems, which triggers rescheduling in order to un-power non-critical spacecraft systems, notify the power system of new load schedules and priorities, and command hardware to power down.

Considerable systems engineering and integration was required to develop and demonstrate autonomous habitat operations. cFS message identifiers required deconfliction both across applications and across multiple flight computers. A tool was created to automatically generate the cFS headers and message identifiers; this tool is called the cFS Command and Data Dictionary (CCDD) and captures a representation of the software components and required messages. The time-triggered scheduler also required knowledge of application run-time, to ensure message send and receive were performed on schedule.

The demonstration involved simulation of the Habitat and multiple sub-systems (power storage and distribution, avionics, and air-side life-support) during a multi-day test at NASA's Johnson Space Center. The foundation of the demonstration was 'quiescent operations' (no maneuvers, EVAs, or docking / undocking operations) of the Habitat during a 55 minute eclipse period. For this demonstration, a Vehicle Systems Manager (VSM) applications managed the spacecraft power distribution system and air-side life support system at a high level of fidelity; additional systems were managed, but with lower fidelity operational constraints and system behavior. Operational constraints for real and simulated loads were derived from on-orbit hardware and future Exploration-capable technology. A total of 13 real and simulated loads were used during the test. Scenarios including both nominal and off-nominal conditions were performed. Over the course of the test, every application performed its desired functions successfully during the simulated tests. The results will inform both future tests, as well as provide insight to NASA's domestic and international partners, as they construct the next generation of space habitats to be used on beyond-Earth missions.

II. Advanced Technology Development

In this section we will describe the technology developed by NASA to autonomously manage a Habitat. The Modular Power System and the associated Autonomous Power Controller are described in Section II.A. The Avionics and Software architecture is described in II.B. Habitat autonomy enabling capabilities are described in Section II.C. Finally, associated Systems Engineering technology developed to manage software configuration is described in II.D.

A. Modular Power System and Autonomous Power Controller

The AES Modular Power System (AMPS) performs power storage and distribution. It consists of two batteries, two MBSUs cross-strapped to two PDUs, each of which feature 8 ports; thus, the system can power 16 independent loads. Loads are not cross-strapped to multiple PDUs. While the broader power system design includes solar arrays and associated power distribution hardware, these elements were unimportant for the purposes of our demonstration. The AMPS hardware configuration is shown in Figure 1; the powered subsystems are further describe in Section III.

The AMPS hardware is managed by the APC [9], which is responsible for ensuring the safe operation of the hardware, including fault management (monitoring and detection of faults), reconfiguring the power system after a fault has been detected, restoring the system to a normal state, forecasting energy availability, and analyzing and implementing user load schedules which prioritize the loads for power. The energy availability forecast consists of total energy and instantaneous power availability constraints for a 2 hour window. The energy availability forecast is generated using a model incorporating energy consumption and production, battery state of charge and state of charge limits, and internal power system faults. Specific battery sizing and demonstration specific operational constraints are described further in Section III.B.

The role of the APC is to safely operate the vehicle's electrical power system and provide power to as many loads as possible. This role can be broken down into two functions, energy management and fault management. The APC is

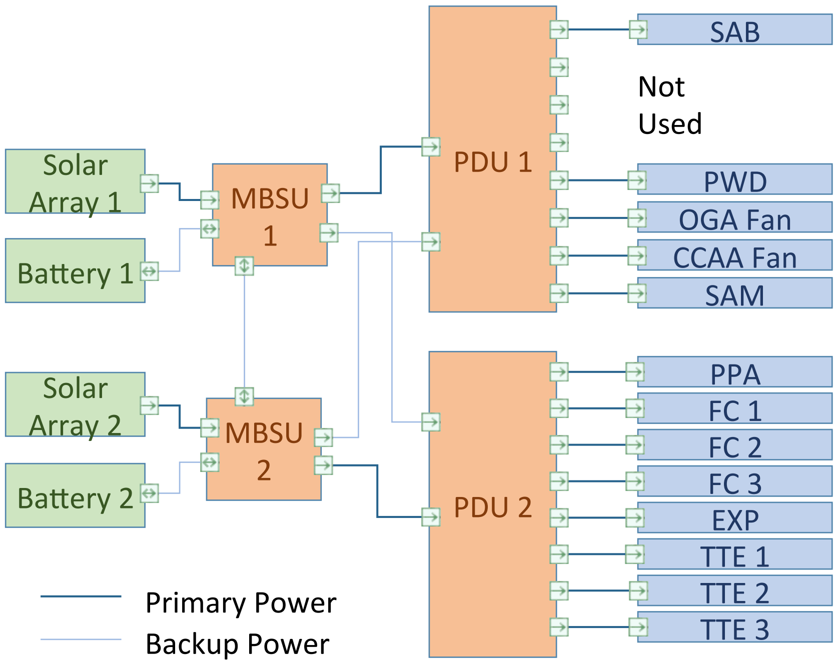


Fig. 1 Power System Configuration showing MBSU-PDU cross-strapping and 8 RPCs, and PDU-MBSU assignment for each load.

responsible for predicting the energy that is available into the future, and ensure that the loads do not consume more energy than is available in the prescribed window and that the power draw does not exceed any of the power system constraints, such as overloading one of the channels. To carry this out, the APC provides the VSM with a 2-hour power profile noting the maximum power per power distribution unit and the total energy over the given time window. The power and energy constraints take into account faults within the electrical power distribution system and ensures the battery state of charge limits are maintained at the end of the power profile. The VSM is free to schedule energy and power usage within those constraints, while respecting the other mission constraints (described in section II.C). The VSM returns to the APC a load schedule, indicating whether a load is powered on or off in a 5 minute time period, and a load shed priority if the load is on, to be used in the event of a fault. This cycle repeats every 5 minutes.

The second function of the APC is to deal with electrical power system faults. The APC monitors the electrical power system for faults within the power system, such as stuck or faulty switches, sensor failures, and short circuits. Once a fault is detected, the APC takes action to reconfigure the electrical power system to deliver power to the highest priority loads; load priority is sent by the VSM as part of its schedule. Next, the APC determines the impact on energy availability and will shed any loads to ensure that the APC does not overdraw from the batteries and keeps the battery state of charge within its limits. Finally, the APC notifies the VSM of the fault, the loads that have been shed, and the new energy availability. If the faulted power system component is removed, or the fault is otherwise resolved, the APC also has the ability to return the new equipment to service and go to its nominal configuration.

B. Avionics and Software

The Avionics and Software foundation of the Habitat is divided into two categories. We first describe the avionics hardware. Then we discuss the software architecture, as well as key applications used in this architecture.

1. Avionics Architecture

The architectural approach used follows that of a Distributed Integrated Modular Architecture (DIMA), where each platform hosts functions of different criticalities [10]. At the core of the architecture are three redundant FCs. Each FC is comprised of a CompactPCI chassis, an Aitech SP0-100 single board computer (PowerQUICC-III MPC8548E), and a TTE A664 network controller. Each FC runs VxWorks 6.9, cFS, and an identical load of cFS software. A fourth chassis performs the role of a Remote Interface Unit (RIU) connecting the FCs to various end devices. Finally, two desktop computers implement some cFS applications and other critical functions (e.g. life support system simulation, display servers). Both are also equipped with TTE network interfaces. The avionics architecture is shown in Figure 4.

Connectivity between platforms is provided by means of three redundant 24-port TTE A664 lab switches. Each of the switches connects to all of the end systems above. Communication between platforms occurs through all redundant switches simultaneously. Messages are forwarded according to static tables. The ARINC 615A protocol is used for loading new tables on the switches over the network. The network cards and switches implement TTE network services in hardware, transparent to the host applications. The network controllers act as Synchronization Masters in the sync protocol, and are assumed to be capable of failing arbitrarily. The switches act as Compression Masters, and are assumed to be constrained to a more restrictive inconsistent omission failure mode [†]. The TTE network is also leveraged for synchronizing flight software on the three FCs and RIU. Depending on the properties of the network scheduling used, it is possible to achieve $<50\mu$s of skew between computers.

The FCs are able to vote redundant TT messages to ensure consistency of shared state and input data. This is necessary to tolerate the asymmetric failure of a TTE card, in which different 'valid' messages could be sent to different receivers (i.e. wrong data, correct cyclic redundancy check). Redundant messages from the same transmission can be grouped in time based on the precision of the network. The voting logic can be implemented in either software (e.g. the end system driver) or hardware (e.g. in an Field Programmable Gate Array (FPGA)) [10]. In either case, a single TT transmission through the three switches is resolved to a single message at the receiver.

All messages generated by the FCs flow through the RIU, which bridges the TTE network with a classical Ethernet Local Area Network (LAN). The Ethernet LAN is connected to a host of distributed subsystem controllers implementing different functions. Each contains an embedded controller responsible for data processing and message distribution. For example, a National Instruments CompactRIO is embedded in each of the AMPS power components. Other systems include communications hardware, human interfaces, and crew displays. In total, the test article features over 15 computer platforms with different processor architectures, byte orders, and operating systems; a subset of all the computers, along with the switches, is shown in Figure 4.

2. Core Flight System

The core Flight System (cFS) is a platform and project independent reusable software framework and set of reusable software applications [11]. cFS was originally developed by NASA Goddard Space Flight Center and now maintained by a NASA Configuration Control Board (CCB). This framework is used as the basis for the flight software for satellite data systems and instruments, but can be used on other embedded systems. cFS is written in C and depends on another software library called the Operating System Abstraction Layer (OSAL). This software is licensed under the NASA Open Source Agreement.

There are three key aspects to the cFS architecture: a dynamic run-time environment, layered software, and a component based design. It is the combination of these key aspects that makes it suitable for reuse on any number of NASA flight projects and/or embedded software systems at a significant cost savings. To support reuse and project independence, the architecture contains a configurable set of requirements and code. The configurable parameters allow the cFS to be tailored for each environment including desk-top and closed loop simulation environments. The typical cFS applications and services, augmented with autonomy applications (discussed in Section II.C), is shown in Figure 2. The cFS architecture simplifies the flight software development process by providing the underlying infrastructure and hosting a runtime environment for development of project/mission specific applications. The cFS architecture also simplifies the flight software maintenance process by providing the ability to change software components during development or in flight without having to restart or reboot the system. The cFS architecture has been proven to reduce time to deploy high quality flight software, facilitate formalized software reuse, and simplify flight software sustaining engineering.

In the remainder of this section we describe select cFS applications developed for this demonstration.

Time-Triggered Ethernet (TTE). The TTE application enables any cFS application to send and receive messages

[†]For more information, see SAE Time-Triggered Ethernet standard AS6802: https://www.sae.org/standards/content/as6802/

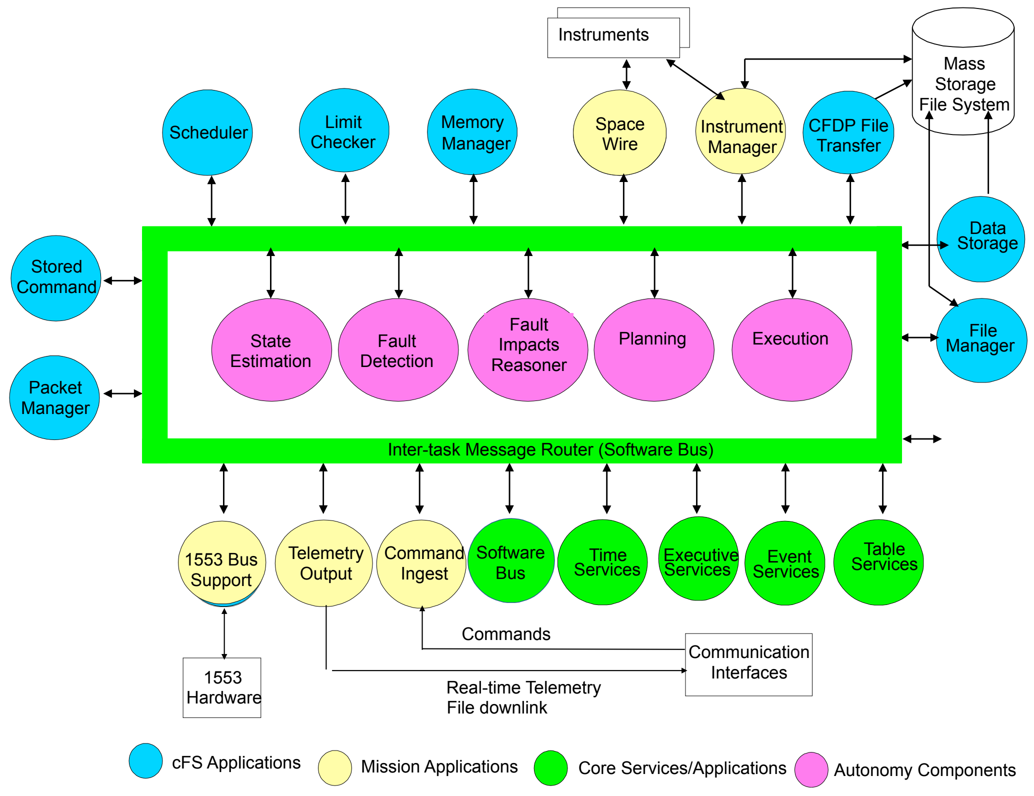


Fig. 2 Core Flight Software showing cFS stock applications, Mission-specific applications, core services, and autonomy enabling applications.

over a TTE network. It provides a simple Application Program Interface (API) by which other apps can claim and access preconfigured data ports. It also enables apps to register interrupts – allowing them to be notified asynchronously when new messages are received, the status of the Ethernet transceivers are changed, or at fixed times in the network schedule. The app is configured using static tables. The tables define the network schedule of the network controller, as well as mapping information for each available TTE port. Unlike most avionics implementations using TTE, the tables can be manipulated at runtime using commands from the ground. Additionally, the app publishes detailed diagnostic information about which ports are available, which tasks own which ports, as well as message error counters and timestamps. This data can be leveraged by onboard fault detection and recovery systems, or used for debugging.

Time-Triggered Scheduler (SCH-TT). In a distributed system, where networked platforms must coordinate to perform some task, it is often necessary to synchronize the execution of the software running on the platforms. This ensures, for example, that data is made available from one platform before it is expected by another. Additionally, it enables fault tolerance approaches in which the same computation is performed on multiple platforms in parallel and then compared. The SCH-TT app leverages the global TTE time base in order to schedule periodic flight software activities at fixed times in the network schedule. Activities scheduled within the same time slot occur simultaneously across all computers in the synchronized set. The flight software schedule is executed within a configurable major frame, which is made to match the cluster cycle of the TTE network. The major frame is divided into multiple minor frames. In each minor frame, the scheduler transmits software bus messages in order to signal other apps to execute or publish telemetry. Moreover, the app uses the concept of "modes" to control the state of the system (e.g. running, paused). Mode change requests sent to one computer are automatically shared synchronously with all its synchronized peers. This design ensures all peers stay synchronized during mode transitions. SCH-TT serves as a drop-in replacement for the standard cFS scheduler, and can therefore be used without impacting other apps. It leverages the TTE cFS application for messaging and interrupt handling.

Software Bus Network (SBN). The cFS was originally intended for simple spacecraft architectures that host the majority of vehicle functions on a single processor. For more distributed architectures, where software runs on multiple platforms, it is necessary for cFS applications to communicate over the network. SBN connects the local software bus on one platform to those of cFS instances running on other platforms. This enables messages published on one bus to be received by subscribers on other buses. This is accomplished by mapping cFS message IDs to network addresses corresponding to different peers. SBN uses a modular architecture in order to support multiple protocols (e.g. TCP, SpaceWire). This design enables SBN to support new protocols through the development of different network plugins. A small interface library, SBN-LIB, allows systems which don't run cFS (such as the APC) to still communicate using the SBN protocol.

SBN was originally designed to be dynamic and flexible. Heartbeat messages can be used to detect other SBN instances over the network, and subscription lists can be updated to change which messages are sent to different peers. However, this approach is not possible when using networks where resources are statically preallocated (e.g. A664-P7, TTE). In such networks, the routing of all traffic flows is decided a priori. To enable the use of these networks, the dynamic aspects of SBN must be disabled. In the case of TTE, static configuration tables are used to map message IDs to the preconfigured TTE ports. These tables are generated in combination with the TTE schedule to ensure they are consistent. They are loaded at initialization, and cannot be changed at runtime.

Level-1 Voter (L1V). While voting redundant TT frames ensures the integrity of messages sent from a single sender, it does not allow receivers to resolve commands from multiple senders. The L1V application is used to resolve multiple software bus messages to a single new message. This capability is useful in cases where a given platform may receive data from multiple redundant devices (e.g. flight computers, sensors). The application is command driven, and can be scheduled to periodically read redundant messages and publish the 'correct' message. Unsuccessful votes, e.g. where a majority is not reached, can be summarized in the form of cFS event messages. The voting algorithms used are modular and configured by cFS tables. Both majority voting and priority-based algorithms are currently implemented. Future work may enable parts of a message to be voted separately, e.g. by majority, averaging, or mid-value selection. To facilitate testing, a separate application was developed to corrupt messages to be voted.

C. Habitat Autonomy Technology

Autonomous operations capabilities include planning and scheduling, plan execution, and fault management. These capabilities are described further in the following subsections.

1. Fault Management

We developed multiple fault management cFS applications with complementary capabilities, applied to different Habitat subsystems.

Fault Detector (FD). The FD processes system data to determine if any defined off-nominal conditions exist. Each off-nominal condition is associated with a test; the inputs to the FD are system data, and the outputs are a set of pass, fail or unknown test results. FD also performs data validation and filtering, such as checks for values that are off-scale, indicating a sensor open or short circuit, and suppresses any test that would check the sensed value against a threshold. Data validation can include checking for sensors that have stopped updating. Since many sensors normally jitter at least a small amount, if a sensor has not changed for many read cycles, it is considered unreliable and any test using the data would be set to unknown until it is updated again. While the data on an invalid sensor reading is suppressed from qualitative testing, the invalid condition itself can be used as an indication of a failure. The sensor itself, or a component involved in converting analog to digital and transmitting the data from sensor to flight computer, may have failed, and the data validity can be a key indicator of these failures. The fault detector is also responsible for assessing system configurations and suppressing tests that are not available or not needed in certain situations, such as a test on pressure output when a pump is turned off as part of normal operations. FD is implemented using the cFS Limit Checker (LC) application, which subscribes to software bus messages and takes action under conditions specified in a table. The usual action is to issue a command or send a message to another application.

Diagnostic Executive (DE). The DE receives test results from FD and packages them to send to the Commercial Off The Shelf (COTS) diagnostic reasoner, called Testability Engineering and Maintenance System (Real Time) (TEAMS-RT) [12]. TEAMS-RT is built into the Diagnostic Executive as a library package, and executes as part of the Diagnostic Executive task in the cFS application set. TEAMS-RT receives the pass, fail and unknown test results that correspond to test points in a diagnostic model. TEAMS-RT correlates test points with failure modes. A failed test can implicate one or more failure modes, while a passing test exonerates failures. With sufficient data, a single failure mode can be identified that is responsible for all failed tests, and an unambiguous failure mode is identified. Otherwise, the reasoner determines the smallest set of possible failure modes that could be responsible for failed tests and presents an ambiguity group of possible failures. If additional data can be obtained, such as by executing procedures or manual tests, it may be possible to resolve the ambiguity. DE operates once per second to provide continually updated diagnostic results that include positive assertions of correct operations, in addition to identification of faults

Fault Impacts Reasoner (FIR). FIR receives failure information from DE and determines the resultant impacts of confirmed failures [13]. Impacts include the loss of function due to a fault, such as the components that have lost electrical power due to a fault in the electrical system. The loss of redundancy due to a fault is also determined. Most critical functions in spacecraft depend on redundancy to assure the availability of the function in spite of failures. Of particular concern is any function that could be lost by a single additional failure, or has become zero-fault tolerant. FIR identifies these changes in redundancy to aid with identifying functions at increased risk, helping operators to determine next worst failures and take mitigation steps to reduce the risks of additional failures, if possible. FIR, coupled with DE, draws a crisp distinction between failed, that is, broken components, from components and functions lost or affected by the failure of a component. The distinction is critical to most effectively take actions to continue safe flight while planning the restoration of capability through reconfiguration or in-flight repairs.

FD, DE and FIR collectively implement a complete fault management capability referred to as Advanced Caution and Warning System (ACAWS) [14]. ACAWS is being adapted to perform fault management for the Orion spacecraft, both for flight controllers and also for crew [13]. We have used the same reasoning technology for this work, but adapted the components for integration with cFS, and also to function without a user interface.

Hybrid Diagnostics Engine (HyDE). HyDE [15] uses hybrid (combined discrete and continuous) models and sensor data from the system being diagnosed to deduce the evolution of the state of the system over time, including changes in state indicative of faults. In contrast with TEAMS, HyDE can represent *hybrid* systems, i.e. mixes of discrete and continuous quantities; TEAMS, by contrast, can only model discrete systems. HyDE models are state transition diagrams, showing how events change the state of the system; these events can be nominal (e.g. commands and processes) or faults. When the sensor data are no longer consistent with the nominal mode transitions, HyDE determines what failure mode or modes are now consistent with the data. While HyDE requires some preprocessing of system sensor data, HyDE can take general inputs (commands, numerical values) from systems, making it more powerful than TEAMS-RT, but more computationally expensive.

2. Vehicle Systems Manager (VSM)

Scheduling and schedule execution are integrated into a single application called the Vehicle Systems Manager (VSM). The job of the VSM is to provide power and load schedules to the APC, listen for faults, and revise the spacecraft-wide mission plan in response to faults, while respecting the mission constraints. We describe the sub-components of this application below.

Planning and scheduling are required to choose and order spacecraft activities in order to meet objectives and satisfy constraints. Planning and scheduling may need to pick from mutually exclusive activities (e.g. there is enough crew time to maintain and repair equipment, or to conduct science activities, but not both). For the purposes of this demonstration, the activities are all known in advance, so it is necessary 'only' to schedule activities. Activities generally have constraints on time, including activity duration, limitations on start and end times, and activity ordering. Activities also use resources; examples include power, thermal, spacecraft memory, and data bandwidth.

For our demonstration, activities correspond to periods of time during which each subsystem, corresponding to a load on the power system, must be used. Most of the spacecraft systems are continuously powered, but some must be switched off periodically. All of the systems use power, and thus energy; some of the subsystems have additional constraints. The specific constraints on scheduling addressed in this work are further described in Section III.B. The most important function of the scheduler is to generate the load schedule to send to the APC. This load schedule must be consistent with the available power and energy for that window as predicted by the APC.

Many spacecraft have the ability to send commands to underlying spacecraft subsystems using a previously developed plan or schedule. These commands are monitored to ensure they completed successfully prior to starting the next activity. Additional monitoring functions can detect and respond to unexpected events and fault codes generated by lower level flight software functions. Plan execution capability can take the form of command sequences or scripts. For our purposes, plan execution operates as a higher level of abstraction, managing many sequences that run simultaneously, while also monitoring for unexpected events. The plan execution system must take as input a new plan, generated by the planner, as the mission continues.

Solving Constrained Integer Programs (SCIP). The VSM scheduler uses the Solving Constrained Integer Programs (SCIP) Optimization tool [16], an open-source tool designed to solve constrained optimization problems which combine mixed-integer, and linear programming, and constraint programming methods. We modeled the power loads as jobs to be scheduled. Each job must then be scheduled subject to periodic constraints to model duty cycles for each load, with constraints requiring some loads power mode to be synchronized with each other, and with constraints on the maximum instantaneous power demand and total energy consumption over the 2-hour plan horizon. The specific constraints are described further in Section III.B. As the mission continues, or when new circumstances (e.g. advancing time, new goals or constraints, unexpected events, or faults) arise, the scheduler generates new plans. The SCIP solver is integrated with cFS to allow re-invocation as new information becomes available.

Plan Execution and Interchange Language (PLEXIL) The plan is executed by the Plan Execution and Interchange Language (PLEXIL), developed as a collaborative effort by NASA and Carnegie Mellon University, and subsequently released as open-source software [17]. PLEXIL is a language for representing flexible robust plans intended to be executed in an uncertain and changing environment. PLEXIL provides well defined execution semantics with contingency handling which can be formally validated and produces deterministic results given the same sequence of environmental inputs. PLEXIL's Execution Engine (executive) executes stored plans written in the PLEXIL language. PLEXIL is responsible for receiving new energy availability messages from the APC and invoking the scheduler, and for sending new load schedules to the APC after generation by the scheduler. PLEXIL is also responsible for receiving fault messages from the fault management functions, and invoking scheduling in the event one or more systems need to be powered off in response to a fault [‡].

D. Systems Engineering and Integration

The cFS Command and Data Dictionary (CCDD) is a tool for managing command and telemetry data descriptions for cFS and its applications, and for producing configuration files for software, ground support, and crew displays. As described earlier, the article under test is a distributed heterogeneous set of avionics computers on a time triggered Ethernet and classic Ethernet network. The vehicle computers, including the voting triplex flight computers and the RIU, run the cFS framework, reusable cFS apps, and custom mission developed software. Each system communicates over the cFS software bus that is extended by the SBN app. Each message produced by each system, whether it is local or shared to its peers, is described in the CCDD database. The CCDD application is driven by an intuitive Webpage interface, shown in Figure 3.

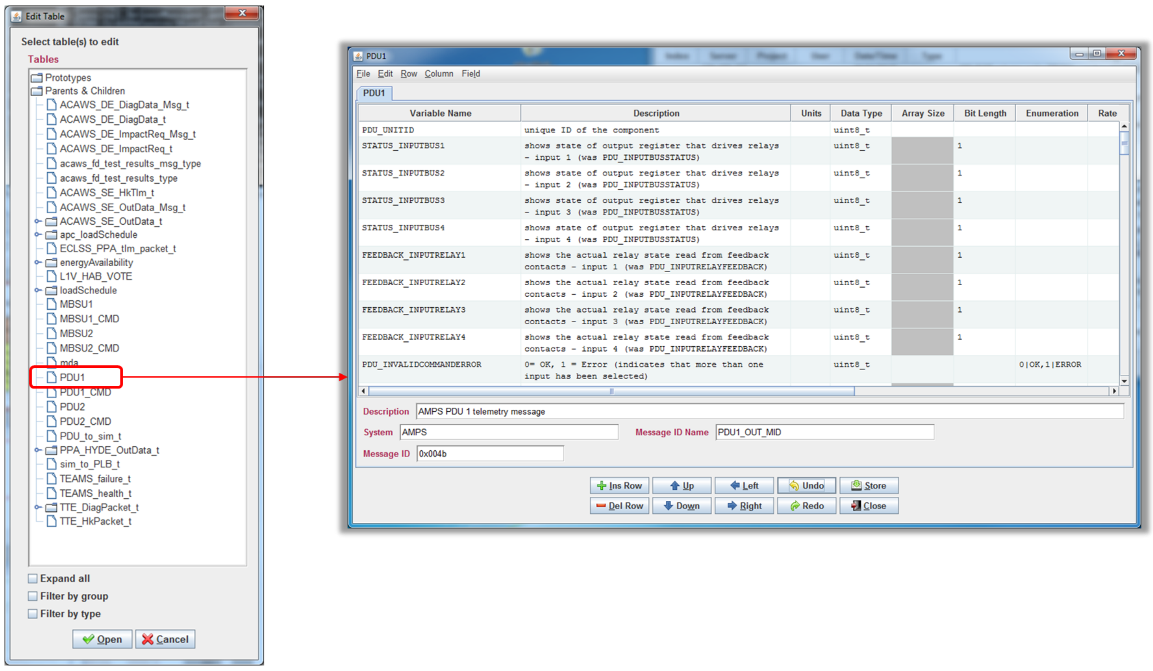
Using a common CCDD database, cFS message identifiers were globally assigned in order to be unique both across applications and across multiple flight computers. Scripts were created to automatically generate the cFS headers and message identifiers for the entire test configuration. The TTE bus also required extensive scheduling, with knowledge of application run-time, to ensure message send and receive was performed on schedule. The TTE network schedule tables are also created based in information captured in the CCDD, and a plugin to generate the schedule is currently being developed.

As described above, habitat autonomy is implemented by a variety of cFS applications. These applications are integrated via cFS messages and commands described in the CCDD process. For example, AMPS generated data enters the fault management application via FD, which uses the cFS LC application; the CCDD captures the description of the messages LC subscribes to. The APC-VSM integration also required numerous cFS messages, which required description using the CCDD process.

Last, we describe the deployment environment for the software under test. The cFS, The TTe and SBN applications run on all computers connected to the TTe swtich; the SBN-LIB enables applications running on the Ethernet switch to communicate. One copy of the SCH-TT is needed on each FC; a final copy of TT-SCH runs on the RIU. The stock SCH application runs on the Linux host. The AMPS software is implemented as several concurrent applications running on multiple computers. The APC is a Linux application running on one host. The data collection applications are connected to a database and data publishing application, APCDBIO, that runs on a second Linux host. This application publishes data via SBN. The FD, DE and HyDE components of the VSM were ported to the triplicate FCs; SCIP, PLEXIL and FIR were run on a host Linux machine.

The complete test article is shown in Figure 4. This figure shows the network and switch configuration, the CPUs, and the subset of cFS applications running on each CPU.

[‡]In more complex simulations, spacecraft subsystems would have command interfaces to power them on or off; in our simulation, loads were controlled entirely by the load plan sent to the APC.



Prototypes
Parents & Children
- ACAWS_DE_DiagData_Msg_t
- ACAWS_DE_DiagData_t
- ACAWS_DE_ImpactReq_Msg_t
- ACAWS_DE_ImpactReq_t
- acaws_fd_test_results_msg_type
- acaws_fd_test_results_type
- ACAWS_SE_HkTlm_t
- ACAWS_SE_OutData_Msg_t
- ACAWS_SE_OutData_t
- apc_loadSchedule
- ECLSS_PPA_tlm_packet_t
- energyAvailability
- L1V_HAB_VOTE
- loadSchedule
- MBSU1
- MBSU1_CMD
- MBSU2
- MBSU2_CMD
- mda
- PDU1
- PDU1_CMD
- PDU2
- PDU2_CMD
- PDU_to_sim_t
- PPA_HYDE_OutData_t
- sim_to_PLB_t
- TEAMS_failure_t
- TEAMS_health_t
- TTE_DiagPacket_t
- TTE_HkPacket_t

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Rate
PDU_UNITID	unique ID of the component		uint8_t				
STATUS_INPUTBUS1	shows state of output register that drives relays - input 1 (was PDU_INPUTBUSSTATUS)		uint8_t		1		
STATUS_INPUTBUS2	shows state of output register that drives relays - input 2 (was PDU_INPUTBUSSTATUS)		uint8_t		1		
STATUS_INPUTBUS3	shows state of output register that drives relays - input 3 (was PDU_INPUTBUSSTATUS)		uint8_t		1		
STATUS_INPUTBUS4	shows state of output register that drives relays - input 4 (was PDU_INPUTBUSSTATUS)		uint8_t		1		
FEEDBACK_INPUTRELAY1	shows the actual relay state read from feedback contacts - input 1 (was PDU_INPUTRELAYFEEDBACK)		uint8_t		1		
FEEDBACK_INPUTRELAY2	shows the actual relay state read from feedback contacts - input 2 (was PDU_INPUTRELAYFEEDBACK)		uint8_t		1		
FEEDBACK_INPUTRELAY3	shows the actual relay state read from feedback contacts - input 3 (was PDU_INPUTRELAYFEEDBACK)		uint8_t		1		
FEEDBACK_INPUTRELAY4	shows the actual relay state read from feedback contacts - input 4 (was PDU_INPUTRELAYFEEDBACK)		uint8_t		1		
PDU_INVALIDCOMMANDERROR	0= OK, 1 = Error (indicates that more than one input has been selected)		uint8_t			0|OK,1|ERROR	

Description: AMPS PDU 1 telemetry message
System: AMPS
Message ID Name: PDU1_OUT_MID
Message ID: 0x004b

Fig. 3 Screenshot of the CCDD. The Web interface allows users to capture the message definition for each subsystem; engines behind the UI produce cFS message definitions and appropriate headers.

E. Testing and Fault Injection

The team developed a variety of infrastructure to facilitate both testing of the system components, and the demonstration itself. Considerable effort in this area was oriented towards fault injection to test the fault management and fault response capabilities of the system.

The cFS Data Storage (DS) application stores cFS message traffic in files. These file are generally stored on a storage device such as a solid state recorder but they could be stored on any file system. Another application must be used in order to transfer the files created by DS to the ground. DS is table-configured and will store specified messages to specified files. Files will automatically be rotated to ensure file size limits are not reached and to more easily support downlink of stored data. In order to facilitate application testing, we developed the DS_Replay (DS_REPLAY) application, which replays messages from DS-generated files at a configurable replay rate, either a fraction of the rate in which the messages were captured by DS or at a set playback frequency. DS_REPLAY is controlled via run-time commands. This allowed us to record test sessions with hardware and replay this data during the software development process.

To ensure adequate power systems fault model coverage, the team built a fault scenario generation tool. This tool permitted the generation and infusion of artificial power systems faults. This was an important way of maturing the fault management application, since silent commanding was only able to produce a limited set of faults. The list of power system faults tested is shown below in Table 1.

Faults were introduced during tests in several different ways. Power systems faults were introduced by implementing a 'silent' cFS command in the AMPS hardware interface. This command would cause a switch in the AMPS hardware to open or close without registering in the normal manner. This unexpected switch change would then be identified both by the APC and, ultimately, ACAWS, as a fault. AMPS faults simulated by silent commanding included Switch Failed Open, Cable Short Circuit, and Internal Power Supply failure. External short circuits were simulated by lowering the trip setpoint below the present current draw, or by raising the current draw of the Programmable Load Bank above the trip threshold. As noted in Section II.B, avionics bus faults, that is, mangled packets, were introduced via a Mangulator application. This application is also a cFS app. An operator can inject some number of mangled packets that must be detected and voted on by the Level 1 voting app. Avionics faults were also accomplished by disconnecting flight computers from the avionics rig. Finally, in some cases, faults are introduced by replaying a dataset containing the appropriate fault signature using the cFS DS_REPLAY application.

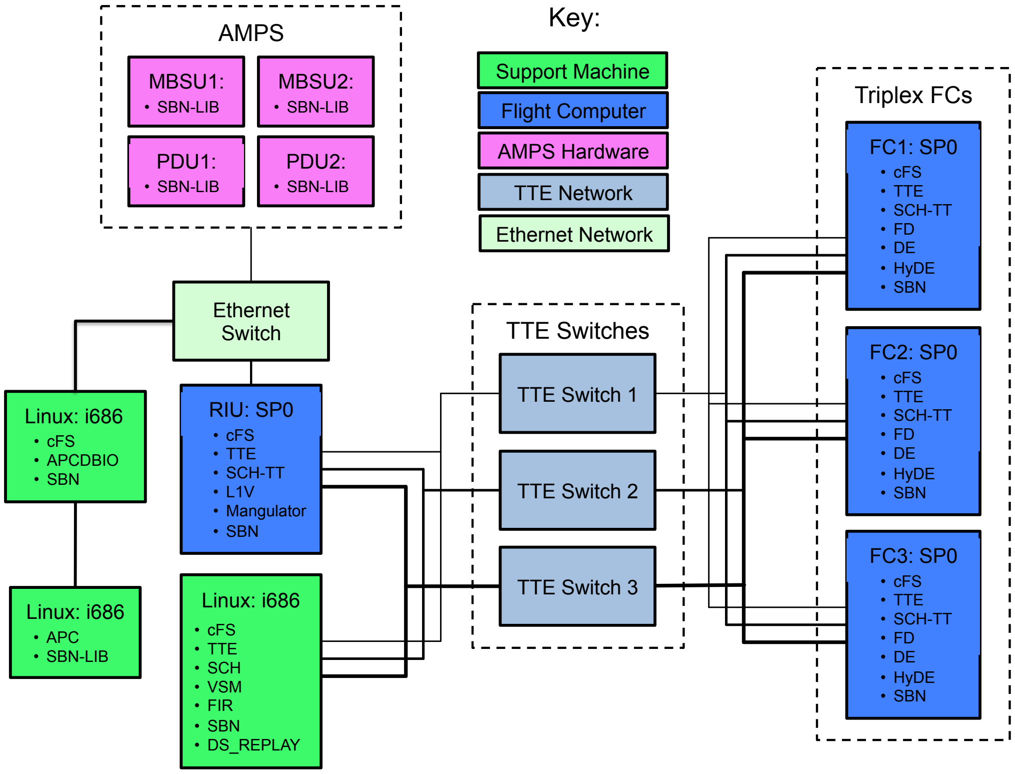

AMPS
MBSU1: • SBN-LIB
MBSU2: • SBN-LIB
PDU1: • SBN-LIB
PDU2: • SBN-LIB
Key:
Support Machine
Flight Computer
AMPS Hardware
TTE Network
Ethernet Network
Triplex FCs
FC1: SP0 • cFS • TTE • SCH-TT • FD • DE • HyDE • SBN
FC2: SP0 • cFS • TTE • SCH-TT • FD • DE • HyDE • SBN
FC3: SP0 • cFS • TTE • SCH-TT • FD • DE • HyDE • SBN
Ethernet Switch
TTE Switches
TTE Switch 1
TTE Switch 2
TTE Switch 3
Linux: i686 • cFS • APCDBIO • SBN
RIU: SP0 • cFS • TTE • SCH-TT • L1V • Mangulator • SBN
Linux: i686 • APC • SBN-LIB
Linux: i686 • cFS • TTE • SCH • VSM • FIR • SBN • DS_REPLAY

Fig. 4 Avionics system architecture with triplicate flight computers, TTE switches, Ethernet backbone, RIU, AMPS hardware, and support computers. This diagram also indicates the cFS configuration for each computer, which is a subset of the application types described in Figure 2.

III. Demonstration of Capability

In this section we first describe the specific elements of the Habitat that were simulated as part of our demonstration. We then describe the operational constraints that must be respected during the demonstration. Finally, we describe the scenarios on which the integrated system was tested.

A. Habitat Simulation

The demonstration was conducted at NASA's Integrated Power, Avionics and Software facility (iPAS). This facility physically hosted the Avionics and Software hardware and the AMPS hardware, as well as support facilities including power and host computers for running simulations of celestial bodies, the Habitat, and other spacecraft functions that were part of other demonstrations. The APC was run at the NASA Glenn Research Center; the APC was integrated with the iPAS over the Space Network Research Federation (SNRF) Network, a NASA internal TCP-IP network.

As described in Section II.A, the number of powered loads attached to the power is constrained by the number of RPCs (8 per PDU, for a total of 16). Three of these were taken up by the three flight computers, and three more were taken up by the TTE switches; the power and energy consumption of these components is known. Seven of the remaining 10 RPCs were occupied by simulated loads representing notional Habitat subsystems. The most complex and interesting simulated loads are the Sabatier reactor (SAB) and the Plasma Pyrolasis Assembly (PPA) [18], which are candidates for a closed-loop air-side Environmental Control and Life Support System (ECLSS). The Sabatier reaction takes as input CO_2 (produced by the crew) and H_2, and produces as output CH_4 (methane) and O_2. ISS hosts a Sabatier reactor; today, the CH_4 is vented to space, resulting in a loss of H_2. The PPA is designed to energize the methane using microwaves to recover some of the H_2, thereby closing the loop on this commodity. For our demonstration, we used

Fault	Number of Instances	Notes
Power Loss	14	Loss of primary power source
Switch Fail Open	30	Switch or relay fails to the open position when intended to be closed
Switch Fail Close	30	Switch or relay fails to the closed position when intended to be open
External Short Circuit	20	Short circuit on a power cable or load
Circuit Overload	14	Overcurrent on a circuit not due to a short
Load Failure	10	General failure of a power load
Internal Power Supply	10	Failure of a Power Supply that provides 28 V power to AMPS internal controllers
Data Corruption	3	Invalid data detected on a data channel
Data Loss	10	Loss of all data on a data channel
Controller Fault	4	Loss of capability to send commands to AMPS components
Bus Short	3	Short circuit on an AMPS module internal bus
Power Card Fault	4	Failure of a PDU power card containing 4 Remote Power Controllers (RPC)

Table 1 Type and number of Modular Power System fault modes.

recorded data of both nominal and off-nominal tests of the PPA and Sabatier. These files were transformed into DS files for replay using the cFS DS_Replay application mentioned above in Section II.D.

The remaining simulated loads were a crew potable water dispenser (PWD), two environmental systems, Oxygen Generation Assembly (OGA) Fan and Cabin Common Air Assembly (CCCA) Fan, Spacecraft Atmosphere Monitor (SAM), and Experiment Facility (EXP). The energy usage values for these systems were created exclusively to drive interesting simulation use cases and outcomes; they are not representative of the energy consumed by either existing or future spacecraft systems. The energy consumption, load shed priority, and power system configuration for each load is described in Table 2. These values were used for every demonstration scenario.

B. Spacecraft Operational Constraints

Load	Power (Kw)	PDU-RPC	Priority	Notes
FC1	0.25	2-2	1	Flight computer (PPC750). Live load. Maximum Power.
FC2	0.25	2-3	2	Flight computer (PPC750). Live load. Maximum Power.
FC3	0.25	2-4	3	Flight computer (PPC750). Live load. Maximum Power.
TTE1	0.045	2-6	4	TTE switch. Live load. Maximum Power.
TTE2	0.045	2-7	5	TTE switch. Live load. Maximum Power.
TTE3	0.045	2-8	6	TTE switch. Live load. Maximum Power.
SAB	0.1	1-1	7	Sabatier reactor (Air-side life support system). Replayed data.
PWD	0.02275	1-5	8	Potable Water Dispenser. Simulated
PPA	0.1	2-1	9	Plasma Pyrolysis Assembly (Air-side life support system). Simulated
EXP	0.2	2-5	10	EXPRESS Rack (Experiment hardware facility). Simulated
OGA Fan	0.3	1-6	11	Oxygen Generator Assembly fan. Simulated
CCCA Fan	0.4	1-7	12	Cryo Cooler Compressor Assembly. Simulated
SAM	0.4	1-8	13	Spaceraft Atmosphere Monitor. Simulated

Table 2 Loads information for each spacecraft subsystem; power, PDU-MBSU assignment, and load shed priority.

The AMPS configuration, created by the team for demonstration purposes, is as follows. Each MBSU and PDU was configured to deliver $2.88Kw$ (nominal) power. The power system, and the battery in particular, is sized assuming a maximum eclipse period of 55 minutes. The spacecraft has 2 batteries; each battery is assumed to be 30 Amp-hour at 126 Volts. This translates to $3.78KwH$ per battery ($W = V \times It$, so $126V \times 30Ah = 3.780\,KWh$). Each battery was

Parameter	Constraint	Notes
Solar Array Power	2.88 Kw	Assumed
MBSU Power	2.88 Kw	Cross-strapped; 1 MBSU may power both PDUs loads
PDU Power	2.88 Kw	No load cross-strapping
Battery DoD	30%	Assumed
Battery Energy	1.134 KwH	Derived from $30Ah$, $126V$
Battery Power Rate	1.235 Kw	Derived from Max Eclipse Dur of 55 mins

Table 3 Power System operational constraints.

constrained to a 30% Depth of Discharge limit (that is, each battery may discharge no more than 30% of its available energy). Thus, the nominal energy available is limited to $3.78 \times 0.3 = 1.134\,KwH$ per battery. In the worst case, we assume this much energy is consumed in the 55 minute eclipse. This translates to a maximum consumption rate of $1.235\,Kw$ per battery during the eclipse. These constraints are summarized in Table 3.

Operational constraints for the demonstration are driven primarily by the power system sizing, loads, and load prioritization. From Table 2, we see that the loads are roughly balanced across PDUs, and initially, MBSUs and batteries: $1.185Kw$ on PDU1, $1.22255Kw$ on PDU2, just under the nominal capacity ($\frac{1.22255KwH \times 55}{60} = 1.12067KwH < 1.134KwH$). Note, however, if a fault occurs, all loads may be served by a single MBSU and, in eclipse, a single battery; in general, loads may need to be shed in the event of a fault during eclipse. A note about battery energy availability and APC behavior: as presently designed, the APC provides the VSM with energy availability, but does not reject a load schedule using more energy than it reported available in the 2 hour window. If the load schedule generated by the VSM does use more energy, the APC will divert energy to recharge the battery, reporting the lower energy available in the next energy availability messages. If the VSM continues to 'borrow', the APC will continue to reduce the energy available (and APC may start shedding more and more loads!) Ultimately, the total available could reach zero. This will also happen if energy usage is higher than predicted due to variation in equipment performance or faults. Neither of these conditions were explored during the demonstrations.

Four of the loads have additional operational constraints. PPA and SAB operating modes are constrained: either both must be powered, or both must be unpowered. We assume CH_4 from Sabatier cannot be stored, so if the PPA is off or faulty, then the Sabatier must be powered off. Similarly, if the Sabatier is off or faulty, there is no reason to use energy to power PPA, so it must be powered off. These two devices also have a 'duty cycle' constraint; they must be powered off for 15 minutes after being powered on for 100 minutes. The PWD also has a 'duty cycle' constraint; it must remain off for 1 minute after being on for 15 minutes. These constraints require the scheduler to periodically power off the SAB, PPA and PWD devices. The EXP is normally powered on continuously. It cannot stay off for more than 30 minutes before it must be powered on again; this constraint is a proxy for a thermal constraint (EXP contains a specimen freezer that cannot be allowed to warm up without losing critical science.) No other devices have any operational constraints, aside from those implicit in the load shed priorities.

C. Demonstration Scenarios

Table 4 describes the scenarios that were used to evaluate the Habitat systems, which we discuss further below.

In the two Nominal scenarios, the environmental conditions are varied (insolation and eclipse) but no faults are injected. The purpose of this scenario is to ensure that all hardware ad software components in the test article functioned properly under nominal conditions. In particular, all subsystem data should be routed to all cFS components correctly, and the Level 1 voter should have nothing to do; the scheduler should ensure all loads remain on except for the PWD, SAB and PPA duty cycle constraints; there should be no false alarms from the fault detection technology.

The 2 FC test and 1 FC test cases demonstrate that the Habitat can remain functioning when the FCs are lost, either due to a malfunctioning FC or loss of power.

The Life Support systems fault involves introducing a PPA fault. In this case, once the fault is detected, the SAB is shut off, the scheduler ensures both systems remain off in the updated schedule, and the VSM notifies the APC of the new load schedule.

The Power Systems Fault is the most complex scenario. This fault eliminates half of the battery energy a few minutes into the eclipse. Specifically, the loss of battery power during an eclipse is due to the failure of the MBSU switch controlling power from the battery. When commanded closed, it failed to the open position, resulting in loss of

Scenario	Description	Notes
Nominal	All systems running, no faults	No eclipse
Nominal plus Eclipse	All systems running, no faults	Eclipse
2 FC Voting Test	1 FC faulted	No eclipse
1 FC Test	2 FCs unpowered	No eclipse
3 FC Voting Test	Inject mangled data	No eclipse
Power Systems Fault	Lose 1 battery during Eclipse	Demonstrate load shed and power replanning
Power System Restored	Reconnect battery	Demonstrate fault cleared and battery recharging
Life Support Fault	PPA Fault	Demonstrate replanning and powering off SAB

Table 4 Demonstration Scenarios.

power from the battery. The battery was still functional, and continued to provide power to the MBSU power supply and hence to the MBSU internal controller, but the failure resulted in complete loss of 120 VDC power input to the MBSU and the ability to use the battery. The timing of this fault ensures that there is insufficient power to keep all of the loads on throughout the eclipse. After the fault, the OGA Fan, CCCA Fan and SAM are shed because they are considered ‘non-essential’ (lowest priority loads in the table). These loads total $1.1Kw$; the remaining loads total $1.30775Kw$. If this fault happens near the beginning of the eclipse, there will not be enough energy in the battery, which only has $1.134KwH$; the remaining loads will use $1.198KwH$ in 55 minutes of operation. The next lowest priority load is EXP; however, the VSM can’t simply shed this load because of the constraint that EXP remain off for less than 30 minutes. However, powering off EXP for the maximum of 30 minutes saves $0.1KwH$, bringing the energy consumed down to $1.098KwH$, which (barely!) avoids draining the sole remaining battery below the 70% state of charge limit. This solution must be found by the VSM application. The EXP can be powered off anytime during the eclipse period for 30 minutes to satisfy this constraint; the scheduler defaults to powering it off as early as possible. The Habitat schedule immediately after the fault, before and after recovery scheduling, is shown in Figure 5. The left figure shows the schedule after the fault but before rescheduling EXP. The fault hits 5 minutes after entering eclipse; the bottom left shows battery energy depletion (green line), until just before exiting eclipse the battery energy drops below the 70% state of charge limit (purple horizontal line, and red vertical bar on right of timeline). After scheduling 30 minutes of EXP off-time (orange block on timeline on right) there is enough energy to last through the eclipse.

This scenario exercises the APC, and the fault management, scheduling and plan execution functions of the VSM. The subsequent clearing of the fault is primarily designed to show that the fault management functions of the APC and VSM will detect resolution of faults and report normal status.

Operational complexity arises when each individual component must address more and more cases consistently and reliably, and when components must interact correctly to address every situation as it arises. While each individual situation is relatively simple for each VSM component to address in isolation, ensuring correct behavior across all cases simultaneously is a challenge. Table 5 shows how operationally complex these scenarios are according to parameters of the simulated systems, operational constraints, fault modes, etc.

System Property	Magnitude	Notes
Power System Data	128 parameters	
Life Support System Data	80 parameters	PPA and Sabatier
Power System Fault Modes	152 faults	
Life Support System Fault Modes	5 faults	Carbon buildup faults, High reflected power, O-ring leak, cracked window
Avionics System Fault Modes	2 faults	Loss of FC, mangled data
Schedule Size	312 steps	(2 hours / 5 schedule minute granularity) x 13 loads
Schedule Constraint Types	6 constraints	Includes power system and load constraints
System commands	60 commands	40 FC-specific commands, 20 ‘global’ commands
System data messages	70 messages	32 FC-specific msgs, 38 ‘global’ msgs

Table 5 Operational Complexity of the Demonstration Scenarios.

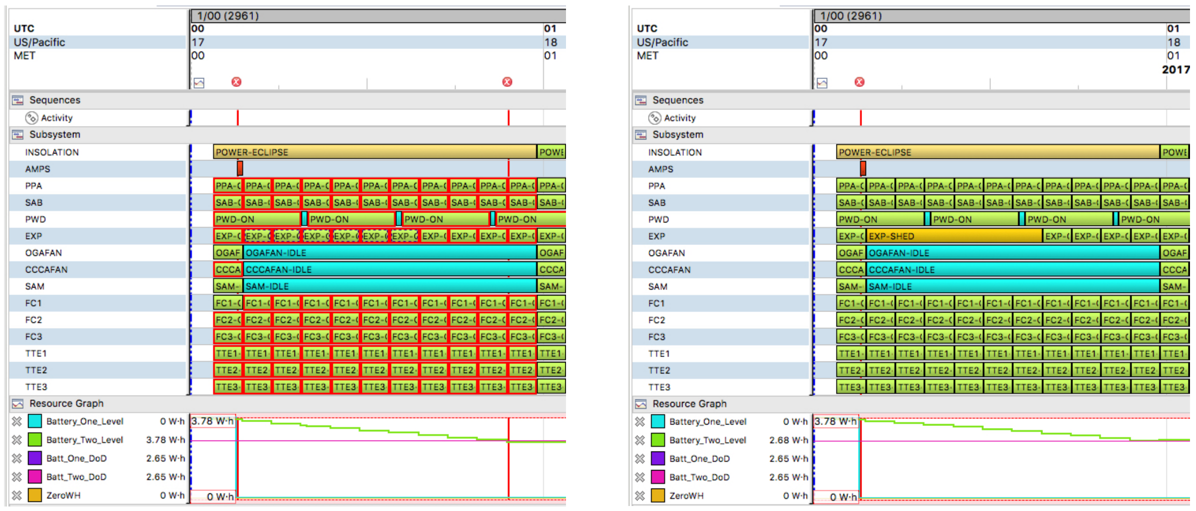


Fig. 5 Power System fault occurrence and recovery timelines. The fault occurs 5 minutes after entering eclipse (left); even after shedding OGA, CCCA and SAM loads, there is insufficient energy to survive eclipse until EXP is powered off for 30 minutes (right).

A note about the number of ‘FC-specific’ commands and messages in the table: a number of cFS apps were running on all three FCs; commands and messages ingested by these applications required an individual identifier, generated by the CCDD process. See discussion in Section II.D.

IV. Lessons Learned and Next Steps

The demonstration scenarios show how to design a habitat that can be autonomously managed by integrating numerous NASA-developed hardware and software technologies. Specifically, the DIMA architecture protects against mangled data and loss of flight computers. The APC notifies other systems of insufficient energy due to faults during eclipse conditions. Finally, the VSM is able to detect and respond to a variety of fault conditions. A solid systems engineering foundation was key to capturing much of the software and hardware system configuration, and automatically generating the cFS message definitions. Systems engineering was also key in building up incremental test infrastructure, specifically, the ability to replay system data and mature VSM software components, as well as introduce faults during the demonstrations.

The successful development and demonstration of these technologies produced many valuable lessons and opportunities for future work. Even though the simulated habitat was small compared to many spacecraft, considerable integration effort was required to ensure message content and semantics was clearly defined. While the CCDD facilitated this integration effort, the technology itself required considerable development effort during the demonstration. Improvements such as ability to generate more and more of the message content and headers are still necessary. The CCDD process can, and should, also capture more cFS app configuration information; examples include some fault management configurations and load energy configuration for the scheduler.

It is notable that many of the VSM components have been developed independently over many years. Considerable work was needed to integrate these technologies with cFS, e.g. making operating system calls OSAL compliant, controlling size of messages, managing big- and little-endian byte order of data on heterogeneous processors, and constructing a schedule of invocations consistent with the TTE-driven schedule developed for the rest of the cFS applications. Enhancing both the CCDD and augmenting SBN to reduce the burden of mixed endianess is a future enhancement. While L1V was demonstrated, switch-level voting was not; this capability will be exercised in a future test. While many of the new applications were demonstrated on the triplicate FC architecture, some components (PLEXIL, SCIP, FIR) were not. PLEXIL has been previously ported to cFSN running on an embedded processor, but this has not been accomplished to date for the other VSM components. Demonstrating all components running on path-to-flight avionics computers is an important and necessary advance. In a similar vein, the power systems software currently consists of many applications running on several desktop computers; consolidating these into a single application is also a key path to flight activity.

The maturation of DS_REPLAY was instrumental in ensuring software testing was possible when hardware was not available. The ability to inject faults is a critical capability to test system behavior in off-nominal conditions. While this capability was available, fault coverage was not complete, and more work is needed to demonstrate behavior of the habitat under a variety of fault conditions.

From the perspective of operational complexity, adding more spacecraft subsystems, increasing the number and complexity of faults, and adding more complex operational dependencies between systems will increase fidelity, as well as impose more requirements on all of the autonomy enabling components. The ECLSS system can be made more realistic by including CO_2 and CH_4 storage and thermal management. The number of faults for PPA and Sabatier vastly exceeds the cases we have data for; more data is forthcoming, which will increase complexity of fault detection. We note that all spacecraft subsystems in our demonstration had only two power consumption levels; on or off. Many real spacecraft systems have different operating modes, e.g. variable speed fans, heaters, and so on. These more complex behaviors will introduce higher operational complexity. Fault response may include high power or energy consuming activities, either by turning on equipment that is turned off, or by placing equipment into high power consuming modes. A final advance is to introduce strategic schedulers or planners that reason for as much as 24 hours into the future, instead of the more limited 2 hour planning horizon demonstrated in our work.

V. Acknowledgements

The authors would like to thank Kevin Seywald for his assistance in reviewing the paper. This work was funded by the NASA Advanced Exploration Systems Program.

References

[1] Crusan, J. C., Smith, R. M., Craig, D. A., Caram, J. M., Guidi, J., Gates, M., Krezel, J. M., and Herrmann, N., "Deep Space Gateway Concept: Extending Human Presence into Cislunar Space." *Proceedings of the IEEE Aerospace Conference*, 2018.

[2] Williams-Byrd, J., Antol, J., Jefferies, S., Goodliff, K., Williams, P., Ambrose, R., Sylvester, A., Anderson, M., Dinsmore, C., Hoffman, S., Lawrence, J., Seiber, M., Schier, J., Frank, J., Alexander, L., Ruff, G., Soeder, J., Guinn, J., and Stafford, M., "Design Considerations for Spacecraft Operations During Uncrewed Dormant Phases of Human Exploration Missions." *Proceedings of the International Astronautical Congress*, 2016.

[3] NASA, "Spaceflight Demonstration of a Power and Propulsion Element (PPE)," `https://www.fbo.gov/spg/NASA/GRC/OPDC20220/80GRC018R0005/listing.html`, 2018. See Amendment 6: Unique Requirements.

[4] Morris, P., Do, M., McCann, R., Spirkovska, L., Schwabacher, M., Frank, J., and Baskaran, V., "Determining Mission Effects of Equipment Failures," *Proceedings of the AIAA Space Conference and Exposition*, 2014.

[5] Frank, J., Iverson, D., Knight, C., Narasimhan, S., Swanson, K., Scott, M., Windrem, M., Pohlkamp, K., Mauldin, J., McGuire, K., and Moses., H., "Demonstrating Autonomous Mission Operations Onboard the International Space Station." *Proceedings of the AIAA Space Conference and Exposition*, 2015.

[6] May, R., Soeder, J. F., Beach, R. F., George, P. J., Frank, J., Schwabacher, M. A., Wang, L., and Lawler, D., "An Architecture to Enable Autonomous Control of Spacecraft," *AIAA Propulsion and Energy Conference*, 2014.

[7] Stetson, H., Frank, J., Haddock, A., Cornelius, R., Wang, L., and Garner., L., "AMO EXPRESS: A Command and Control Experiment for Crew Autonomy." *Proceedings of the AIAA Space Conference and Exposition*, 2015.

[8] Aaseng, G., and Frank., J., "Transitioning Autonomous Systems Technology Research to a Flight Software Environment." *Proceedings of the AIAA Conference on Space Operations*, 2016.

[9] Csank, J., Soeder, J., Follo, J., Muscatello, M., Hau, Y. H., and Carbone, M., "An Intelligent Autonomous Power Controller for the NASA Human Deep Space Gateway," *Proceedings of the AIAA International Energy Conversion Engineering Conference*, 2018.

[10] Loveless, A., "On TT Ethernet for Integrated Fault-Tolerant Spacecraft Networks." *Proceedings of the AIAA Space Conference and Exposition*, 2015.

[11] McComas, D., Wilmot, J., and Cudmore, A., "The Core Flight System (cFS) Community: Providing Low Cost Solutions for Small Spacecraft," *Proceedings of the* 30th *AIAA /USU Conference on Small Satellites*, 2016.

[12] Mathur, A., Deb, S., and Pattipati, K., "Modeling and Real-Time Diagnostics in TEAMS-RT," *Proceedings of the American Control Conference*, 1998.

[13] Aaseng, G., Barszcz, E., Valdez, H., and Moses, H., "Scaling Up Model-Based Diagnostic and Fault Effects Reasoning for Spacecraft," *Proceedings of the AIAA Conference on Space Operations*, 2015.

[14] McCann, R., Spirkovska, L., and Smith, I., "Putting ISHM Capabilities to Work: Development of an Advanced Caution and Warning System for Crewed Spacecraft." *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, 2013.

[15] Narasimham, S., and Brownstone, L., "HyDE - A General Framework for Stochastic and Hybrid Model - Based Diagnosis," *Proceedings of the 18th International Workshop on the Principles and Practices of Diagnosis*, 2007, pp. 162 – 169.

[16] Achterberg., T., "SCIP: solving constraint integer programs." *Math. Prog. Comp.*, Vol. 1, 2009, pp. 1 – 41.

[17] Verma, V., Jónsson, A., Pasareanu, C., and Iatauro, M., "Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations," *Proceedings of the AIAA Space Conference*, 2006.

[18] Greenwood, Z., Abney, M., Perry, J., Miller, L., Dahl, R., Hadley, N., Wambolt, S., and Wheeler, R., "Increased Oxygen Recovery from Sabatier Systems Using Plasma Pyrolysis Technology and Metal Hydride Separation," *Proceedings of the* 45^{th} *AIAA International Conference on Environmental Systems*, 2015.